Experiments in Context-Sensitive Incremental and Modular Static Analysis in CiaoPP
(Extended Abstract)

Isabel Garcia-Contreras[1,2], Jose F. Morales[1], and Manuel V. Hermenegildo[1,2]

[1] IMDEA Software Institute
{isabel.garcia, josef.morales, manuel.hermenegildo}@imdea.org
[2] Universidad Politécnica de Madrid

Keywords: Program Analysis · Abstract Interpretation · Fixpoint Algorithms · Incremental Analysis · Modular Analysis · Logic Programming

Introduction and motivation. Abstract interpretation is a widely used technique for automatically detecting errors and proving program properties related to correctness, security, cost, etc. Performing such analysis during software development helps in early bug detection, but, given the size and complex structure of real-life programs, triggering a complete reanalysis for each set of changes is often too costly. However, development iterations normally involve small modifications in practice, which are often isolated within a small number of files or components. This can be taken advantage of to reduce the cost of re-analysis by reusing previous information. In particular, in CiaoPP [4, 2], incrementality-based cost reductions have been achieved to date at two levels: on one hand, ***modular* context-sensitive analysis** has been used to obtain global information on the whole program by iterating over local analyses of its components (modules). While this technique has been primarily aimed at reducing memory footprint, it can also achieve some incrementality. On the other hand, **fine grain context-sensitive *incremental* analysis** [3] identifies, invalidates, and recomputes only those parts of the analysis results that are affected by program changes. This analysis has been used to achieve very high levels of incrementality, with finer granularity (e.g., at program line level), but it does not take advantage of the module structure.

Objectives. We have recently been working on combining these two techniques within CiaoPP in order to achieve incrementality both at the intra- and inter-modular levels. Extending the *context-sensitive*, fine-grained, incremental analysis techniques to the modular setting requires dealing with the fact that the analysis of a module depends on the analysis of other modules in complex ways, through several paths to different versions (summaries) of the procedures. In order to bridge this gap we have developed a framework that analyzes separately the modules of a modular program, using context-sensitive fixpoint analysis while achieving both inter-modular (coarse-grain) and intra-modular (fine-grain) incrementality. Our objective is to give an overview (and demo) of the approach and, specially, report on the results (i.e., incremental gains) obtained so far.

Algorithm. The essence of the algorithm is that the concrete (possibly infinite) program execution trees are abstracted as graphs (essentially, regular trees), with the analysis information split by procedure (predicate), and partitioned by module, while tracking the dependencies between predicates and modules. We solve the problems related to the propagation of the fine-grain change information across module

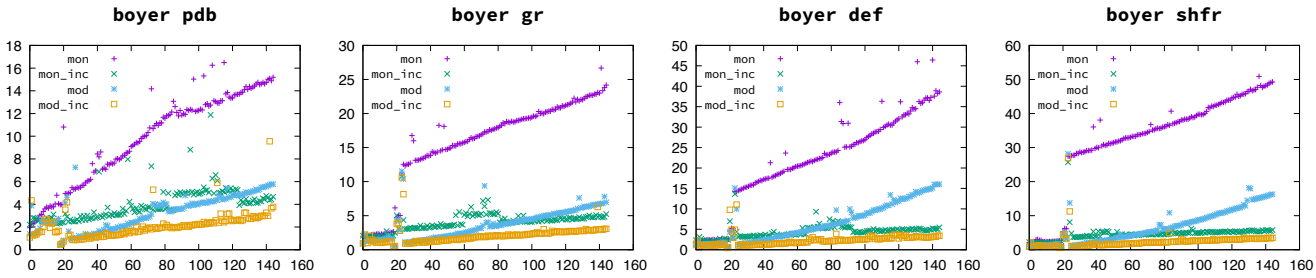

Fig. 1. Incremental analysis with different settings and domains

boundaries. We also work out the actions needed to recompute the analysis incrementally after multiple additions and deletions across modules in the program. We show that the analysis result is always correct and it is the best (most accurate) over-approximation of the actual behavior of the program. The full description of the algorithm is given in [1].

Experiments. The results of our experiments are also detailed in [1] and can be reproduced in `https://github.com/ciao-lang/ciaopp_tests`, but we include here some results for one characteristic program: a stylized version of the `Boyer-Moore` theorem prover, written modularly in Prolog. The experiment chosen consists in adding all the predicates in all the modules in the program one clause at a time, (re)running the analysis after each addition, as a stress test of the algorithm. The experiment is run with different domains: reachability (`pdb`), groundess (`gr`), dependency tracking via propositional clauses (`def`), and sharing and freeness, (pointer sharing and uninitialized pointers, `shfr`). The results (in terms of CiaoPP analysis time) are given in Fig. 1. The analyzer settings shown are the traditional monolithic, non-incremental analysis (`mon`), the (monolithic) incremental algorithm (`mon_inc`), the modular (coarse-grain incremental) algorithm (`mod`) and the current, fine-grain modular algorithm (`mod_inc`). It can be seen that the modular incremental approach is faster in most cases, and all incremental approaches are significantly faster than the traditional, non-incremental approach. In spite of the different costs of the various abstract domains (e.g., analyzing with `def` takes at most $40ms$ while analyzing with `pdb` takes at most $17ms$), the trend is clear: incremental analysis performs better than running the analysis from scratch. In general, the implementation and evaluation show that the proposed modular algorithm achieves competitive and, in some cases, improved, performance when compared to existing non-modular, fine-grain incremental analysis techniques. Furthermore, thanks to the more detailed propagation of inter-modular analysis information, our new algorithm outperforms the traditional modular analysis even when analyzing from scratch.

References

1. Garcia-Contreras, I., Morales, J.F., Hermenegildo, M.V.: An Approach to Incremental and Modular Context-sensitive Analysis. Tech. Rep. 1804.01839 (v4), arXiv (July 2019)
2. Hermenegildo, M., Puebla, G., Bueno, F., Lopez-Garcia, P.: Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor). Science of Comp. Progr. **58**(1–2) (2005)
3. Hermenegildo, M.V., Puebla, G., Marriott, K., Stuckey, P.: Incremental Analysis of Constraint Logic Programs. ACM TOPLAS **22**(2), 187–223 (March 2000)
4. Hermenegildo, M., Bueno, F., Carro, M., Lopez-Garcia, P., Mera, E., Morales, J., Puebla, G.: An Overview of Ciao and its Design Philosophy. TPLP **12**(1–2), 219–252 (2012)